Rapid and Automated Urban Modeling Techniques for UAS Applications

Youngjun Choi[1], David Pate[2], Simon Briceno[3] and Dimitri N. Mavris[4]

***Abstract*— Urban models for testing UAV path-planning algorithms commonly apply simple representations using cuboid or cylinderical shapes which may not capture the characteristics of a urban environment. To address this limitation of existing urban models, this paper presents two urban modeling techniques for an unmanned aircraft flight simulation in an urban environment. The first proposed urban modeling technique is an airborne LiDAR source-based approach that incorporates machine learning algorithms to identify the number of buildings and characterize them from the LiDAR information. The second proposed urban modeling technique is an artificial urban modeling technique without any airborne LiDAR resources that applies an adaptive spacing method, an iterative algorithm to define an artificial urban environment. Unlike the LiDAR source-based approach that creates an approximated urban model, the adaptive spacing-based urban modeling algorithm generates an artificial urban environment that is visually different from a reference city, but has similar the characteristics to it. To demonstrate the two proposed urban modeling techniques, numerical simulations are conducted using open-source datasets to construct several realistic urban models.**

I. Introduction

The operation of Unmanned Aircraft Systems (UAS) in low-altitude airspace, especially in urban environments, has become an essential research field due to the many critical missions UAS can perform. For example, UAS can be used for transportation infrastructure maintenance, goods delivery, surveying and mapping (e.g., infrastructure, topography, digital terrain modeling), precision agriculture, and emergency response. When UAS operates in urban environments, a collision avoidance trajectory is used to ensure the aircraft do not come too close to buildings or other objects. These algorithms are often tested by numerical simulation. For a realistic simulation, an appropriate representation of an urban environment is needed for a realistic simulation.

For a flight control simulation, the urban model commonly describes buildings through a priori knowledge about building shapes, such as a cuboid or cylindrical. The on-board sensor in the flight controller receives building information from the urban model based on the sensor FOV (Field of View) specification and the vehicle state. The received building information is applied to update a collision avoidance algorithm for the upcoming obstacle avoidance. For example, Xu et al. employed a p-norm based model to describe cuboid-shaped obstacles, such as tall buildings, to test a helicopter obstacle avoidance algorithm [1]. Their simulation environment includes three buildings, but our model of interest has a more dense building environment. In the Wen et al. collision avoidance research [2], a low-altitude obstacle model is incorporated in a real-time path-planning algorithm. The obstacle model comprises multiple cuboids, but they do not explain that the obstacle environment is relevant to the realistic obstacle environment. The urban model written by Griffiths et al. has cubic and cylindrical buildings of an urban model for obstacle and terrain avoidance of UAVs [3]. However, this paper also does not present the relationship between the generated urban model and the actual urban environment. Stastny et al. generates an urban model that is Midtown Manhattan, New York [4]. The urban model includes three simplified building models to verify their collision avoidance algorithm, and shows the relationship between a simulation urban model and real Manhattan buildings. However, despite having a representative building model, the paper does not refer to the urban modeling process.

Urban modeling is also an active domain of research within the field of computer science. The traditional approach to constructing an urban model is based on aerial images. This image- and data-based approach is a computationally expensive and time consuming manual process. To overcome these difficulties, airborne LiDAR (Light Detection and Ranging) sensors (or instruments) that collect high fidelity information with centimeter precision are used. The drawbacks of LiDAR data include uncertainties from inertial navigation errors, sensor noise, and reflection errors from ground object surfaces. These uncertainties degrade the quality of an urban model. On the other hand, unlike the image-based urban modeling approach, a user can easily handle the LiDAR sources to generate an urban model because the LiDAR information consists of a point cloud with geometry information. Therefore, additional image processing is not required to obtain urban geometry information. Due to these advantages, LiDAR sensors are often used to provide data for the generation of a realistic urban model [5], [6], [7], [8]. You et al.[7] devised an automated process using LiDAR information. This approach reconstructs LiDAR information through re-sampling, hole filling, and tessellation. Then, the reconstructed information is provided to a classification algorithm to discriminate between buildings and bare ground. The classified result is refined by building primitives, and then the refined information is optimized by fitting and filtering

[1]Research Engineer, School of Aerospace Engineering, Georgia Institute of Technology, ychoi95@gatech.edu

[2]Research Engineer, Georgia Tech Research Institute, david.pate@gtri.gatech.edu

[3]Senior Research Engineer, School of Aerospace Engineering, Georgia Institute of Technology, briceno@gatech.edu

[4]S.P. Langley Distinguished Regents Professor, School of Aerospace Engineering, Georgia Institute of Technology, dimitri.mavris@aerospace.gatech.edu

techniques. Based on this automated process, Hu et al. [6] suggested an advanced urban modeling method combining airborne LiDAR data and aerial imagery information. This additional imagery information allows more precise edge detection and improves computational complexity. Zhou et al. [8] also introduced a building modeling method that includes a classification algorithm to distinguish between vegetation area and building area, a roof generation algorithm from boundary detection, and creates polygon meshes to construct a building model.

These urban modeling approaches in the computer science domain may not be applicable to the urban modeling of a UAS obstacle avoidance problem, since these methods include detailed building/vegetation models, which are unnecessary information. In addition, the polygon-based building models require high computational processing time in a forward-looking LiDAR sensor model that collects point cloud information from rays within its field of view. In the sensor model, excessive numbers of polygons require more computational resources to calculate cross sections between the polygons and the rays for point-cloud information. Depending on the UAV operation problem, an urban environment may not require a detailed description near the ground because the UAVs are required to fly above a minimum altitude for operational safety.

This paper introduces a new rapid, data-driven, and grid-based urban modeling methodology to generate a realistic urban environment for a UAS flight simulation, specifically, a virtual urban environment to test an obstacle avoidance algorithm. This urban modeling could be possibly utilized for urban operational research, such as on-demand mobility and urban package delivery. This methodology entails five steps: resampling/refining data, classification, principal component analysis, grid generation, and building construction. The result of the urban modeling of San Diego has been successfully implemented to test a UAV obstacle avoidance algorithm [9][10].

Unfortunately, aerial LiDAR data may not be available for every city of interest, and collecting this data can be very expensive. In UAS flight simulations, diverse and realistic urban models are key to check the robustness of UAS obstacle avoidance algorithms. To address this need for generating more urban models, this paper also introduces an alternative urban modeling approach that does not rely on any airborne LiDAR resources and easily constructs an artificial urban model.

In the remainder of the paper, we introduces the new rapid, data-driven, and grid-based urban modeling methodology, and shows several demonstrations with different cities. This paper introduces the artificial urban modeling method, explains the details of the steps through a toy problem and demonstrates this method through the San Diego model. After that, this paper states concluding remarks.

II. Rapid, data-driven, and grid-based urban modeling

An urban model must balance the detailed representation of the environment with computational cost, which depends on the complexity of the urban model. For the purpose of the simulation, the urban model does not need to be overly detailed; items such as bridges, structures related to electric power, and other infrastructure can be excluded.

To address the objectives of the urban modeling, we propose a new rapid, data-driven, and grid-based urban modeling method. The proposed method utilizes airborne LiDAR sources that comprises five steps. The first step includes collecting LiDAR data, resampling, and refining the original data to reduce the amount of LiDAR information. The second step is solving a clustering problem for the identification of individual building components from unlabeled LiDAR data. The third step specifies the principal directions of each building to define its rotational angle. The fourth step is grid generation that defines the fidelity of an urban model. The final step is urban generation, which entails defining the width, length, and height of all buildings. The following subsections describe the details of each step.

A. Collecting/down-selecting/refining LiDAR data

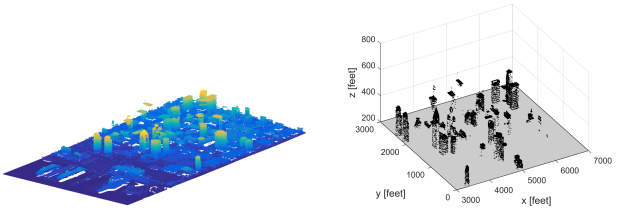
The first step is collecting urban information from airborne LiDAR sources and down-selecting the collected data. The Open-source airborne LiDAR information is available from multiple organizations: United States Interagency Elevation Inventory (USIEI)[1], Open-Topography[2], USGS Earth Explore[3], PAMAP LiDAR Elevation Data[4], and Indiana Spatial Data Portal[5]. In this paper, the LiDAR resources from Open-Topography and PAMAP LiDAR Elevation Data are employed for example case studies to test the proposed urban modeling methodology. Open-Topography is a web-based open-source database with high-resolution topographic data. PAMAP is also a digital LiDAR database of Pennsylvania, which is managed by the Pennsylvania Department of Conservation and Natural Resources and the Pennsylvania Bureau of Topographic & Geologic survey. The collected LiDAR data is converted to a readable format using Rapidlasso tool, an open-source post-processing program [6].

We select the downtown region of San Diego, CA presented in Figure 1(a) as a working example. The size of the selected area is approximately 6000 (feet) by 3000 (feet). To eliminate unnecessary point cloud information, an altitude constraint is also considered. The reason of the altitude constraint is that potential 'UAS operation scenarios in an urban environment most high-speed transit in an urban environment is not near the ground [11]. Moreover, many objects near the ground exist, such as vegetation and complex ground facilities that may increase the complexity of an

[1]https: //coast.noaa.gov/inventory
[2]https: //www.opentopography.org
[3]https: //earthexplorer.usgs.gov
[4]https: //www.dcnr.state.pa.us/topogeo/pamap/lidar/index.htm
[5]https: //gis.iu.edu/datasetInfo/statewide/in_2011.php
[6]https: //www.rapidlasso.com

urban model. Hence, this elimination of the point cloud near the ground improves the computational speed to generate an urban model by reducing the amount of point cloud information.

In an example study, we assume that the altitude constraint is 200 (feet). According to UAS NASA project document, 'Unmanned Aircraft System (UAS) Traffic Management (UTM)', they proposed an UAS operation airspace for high-speed transit drones between 200 (feet) and 400 (feet), and a no-fly zone between 400 (feet) and 500 (feet) [11]. They also define the airspace above 500 (feet) that is shared with manned aircraft as the integrated airspace. The result of the altitude constraint is 40,293 points presented in Figure 1(b), which is approximately 1.60 percent of the original raw LiDAR information.



(a) Raw LiDAR data (b) Resampled point cloud

Fig. 1: Result of resampled point cloud in a San Diego example

B. Identification of building clusters

The second step is determining individual building information from the resampled point cloud. Since the resampled point cloud is unlabeled, to characterize individual building dimensions, the data points need to be partitioned into clusters corresponding to individual buildings.

The clustering algorithm, which is an unsupervised learning technique in the machine learning field, is a popular technique to solve unlabeled problems such as clustering and anomaly detection. The notable classical approaches are the k-means and the k-medoids algorithms [12]. The main notable weakness of these clustering methods is defining the optimal number of clusters, k, a priori (See [13], [14]). Another clustering method is the partitioning method using algebraic graph theory that divides points into groups based on similarity characteristics. The classical graph-based partitioning methods apply diverse methods such as normalized cut, ratio cut, or graph cut with an optimization framework. However, these classical graph-based methods are highly computationally complex, considered an NP-hard problem [14]. A spectral clustering method that is computationally efficient is an alternative graph-based approach. A notable spectral clustering algorithm is the NJW (Andrew Y. Ng, Michael I. Jordan, and Yair Weiss) algorithm proposed by Ng. et al [15]. This NJW algorithm solves the k-means algorithm based on the eigenvector with top n eigenvalues of the Laplacian matrix $\mathbf{L}$. The Laplacian matrix $\mathbf{L}$ is computed from the affinity matrix A defined by a similarity measure. One common similarity measure is the Gaussian function. The NJW algorithm, however, needs to specify the optimal top n, and additionally define the optimal number of clusters k a priori.

The density-based spatial clustering of applications with noise (DBSCAN) method suggested by Ester et al. [16], determines groups of points based on the user defined minimum density. The robustness of the DBSCAN technique is controlled by considering the maximum radius of a neighborhood ϵ and the minimum number of points p in the group to satisfy the maximum radius. The point density approach is an ideal structure to group adjacent points and can also eliminate noise data. This technique enables us to solve the non-linear clustering problem and provide a robust solution against uncertainties. Therefore, we opt to use the DBSCAN technique on the clustering problem from the resampled LiDAR point cloud.

Figure 2 is the clustering result from the DBSCAN technique. The parameters in the DBSCAN, the maximum radius of a neighborhood ϵ and the minimum number of points p, are chosen to be 50 feet and 50 points, respectively. We note that these two parameters are determined by observing characteristics based on the trade-off studies to capture precisely individual clusters. We also note that if these parameters are small values, the result is likely to have many clusters. On the other hand, if the variables are set to high values, the result is likely to have fewer clusters. This particular case results in 37 identified clusters.

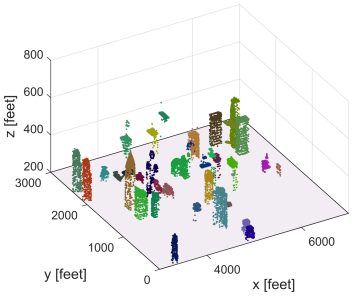


Fig. 2: Clustering result using the DBSCAN technique

C. Identification of rotational angle and construction of a building

The third step is to identify the rotational angle of an individual cluster using the clustering result. The rotational angle produces a more accurate estimation of a building's length-/width/height in two local major axes of each building. To be more specific, the result of the building always determined based on the global axis regardless of the rotational angle. Figure 3 presents the example of the issue, not considering the rotational angle. The length and height are evaluated by maximum/minimum values of point cloud, which is the red line in the figure. The result of the length/height estimation is not accurate. To obtain a more precise characterization of the building, we should also consider the rotational angle of the point cloud information.

The rotational angle of each cluster can be specified by Principal Component Analysis (PCA) that provides dominant direction from given point cloud information. In the

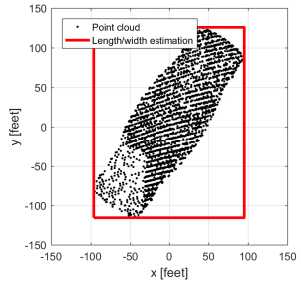


Fig. 3: Building shape without considering rotational angle

mathematical context, the PCA technique is an orthogonal linear transformation method that changes the original coordinate system into a new orthogonal coordinate system with the highest variance. This PCA technique has been widely implemented in various fields such as pattern recognition, compressing data structure, and reduction of dimensions minimizing the loss of the data information [17] [18] [19]. This paper will briefly discuss an overall concept of principal component analysis based on the reference book written by Jolliffe [20].

It is assumed that data D are $D = \{\mathbf{x}_1,\ \mathbf{x}_2,\ \cdots,\ \mathbf{x}_m\}$. From the given data, we can compute the mean and covariance according to the following equations:

$$\mu = E[\mathbf{x}] = \frac{1}{n}\sum_{i=1}^{m}\mathbf{x}_i \tag{1}$$

$$\Sigma = E[(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T] \tag{2}$$

The covariance matrix is represented by a linear transformation equation

$$\Sigma = \mathbf{A}\Lambda\mathbf{A}^T, \tag{3}$$

where $\mathbf{A}$ is the orthogonal linear transformation matrix of eigenvectors, and Λ is the diagonal matrix that includes eigenvalues $\Lambda = \{\lambda_1,\ \lambda_2,\ ...,\ \lambda_n\}$. The covariance Σ can also be written, $\Sigma = \lambda_1 a_1^T a_1 + \lambda_2 a_2^T a_2 + \cdots + \lambda_n a_n^T a_n$. Reducing the dimension of the given data can be defined by introducing a transformation in a latent space. We select a new transformation matrix $\mathbf{A}_k = \{a_1,\ a_2,\ \cdots,\ a_k\} \in \mathbb{R}^{n\times k}$. The k eigenvectors, where $k \leq m$, are specified from the first k largest eigenvalues, $\lambda_1 \geq \lambda_2 \geq \cdots \geq 0$. The physical meaning of this process is that we select a k-dimensional latent space with the k largest variances. The covariance matrix can be represented by a linear transformation equation

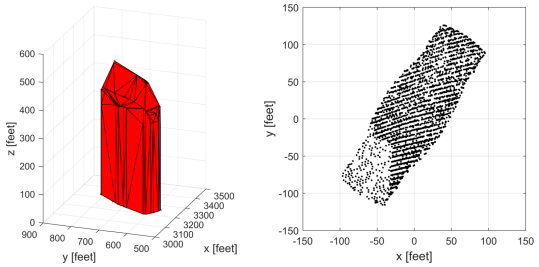
$$\nu = A_k^T\mathbf{X}, \tag{4}$$

where the k-dimensional orthogonal transformation matrix is $A_k \in \mathbb{R}^{k\times n}$ and the result of the latent variable is $\nu \in \mathbb{R}^{n\times k}$. The inverse transformation onto the original space can be defined as

$$\mathbf{X} = A_k\nu. \tag{5}$$

Using the PCA technique, we compute the rotational angle of each cluster. As a demonstration, we applied this PCA algorithm to the group of data points representing the Manchester Grand Hyatt San Diego building shown in Figure 4(a).

Since the rotational angle is defined in the z-axis of the global coordinate system, all the points of the example cluster are projected onto a ground plane presented in Figure 4(b), $\bar{\mathbf{x}} \in R^2$. Using the projected point cloud, the corresponding eigenvalues and eigenvectors are estimated. The physical meaning of computing eigenvalues is to indicate the variance of the point cloud information, and eigenvectors describe the rotation angle. The computed two eigenvectors are shown in Figure 5.



(a) 3D model (b) Raw point cloud information

Fig. 4: Point cloud of Manchester Grand Hyatt San Diego

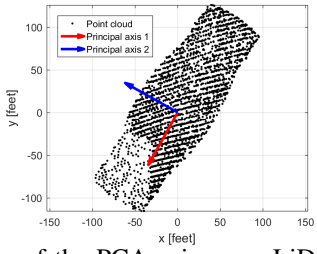
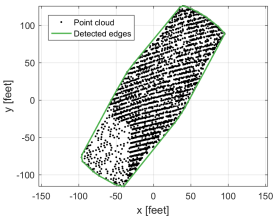
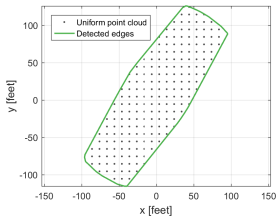


Fig. 5: Problem of the PCA using raw LiDAR information

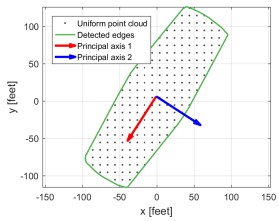
The visual inspection of the PCA result reveals that the actual computed two principal axes are slightly shifted from the axes with the most variance. The reason is for these shifted principal axes is that the given point cloud information has some irregular patterns. To be more specific, the point cloud on the bottom of the left side is more sparse than the point cloud in the top of the right side. To solve this, some additional steps are proposed in this paper. The main idea of the additional steps is using uniformly spread point cloud data instead of the irregular point cloud information to minimize bias resulting from the irregular point density. The additional process of using the uniformly spread points includes boundary points detection, addition of a uniformly spread point cloud, and analysis of the principal axes. Figure 6 presents the results of the suggested PCA process. The result visually presents that the estimated principal axes detect the principal components more precisely. From this result, we can conclude that the modified PCA algorithm helps reduce the biased rotation caused by the irregular point density. From the PCA result, we can transform the point cloud data of the selected cluster into the principal axes using Equation 4. The transformed point data in the principal coordinate system is presented in Figure 7.



(a) Boundary points detection



(b) Including uniformly spreaded point cloud



(c) Result of the principal axes

Fig. 6: Modified PCA approach

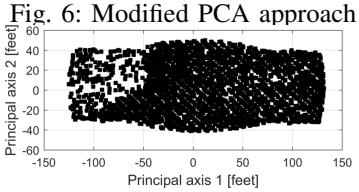
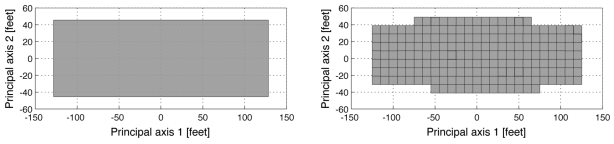


Fig. 7: Point cloud data in the principal coordinate system

D. Grid generation

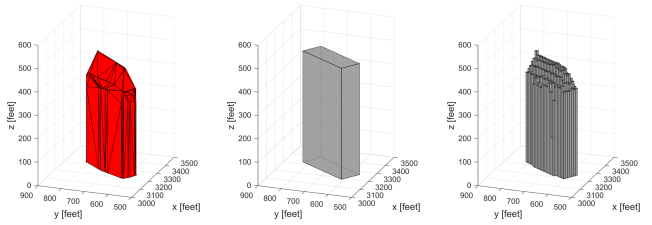
The next step is to characterize a composite rectangular cuboid building configuration. It identifies width, length, and height based on the transformed point cloud information. In this step, we introduce a grid generation that enables us to adjust the fidelity of a building. The grid generation controls the resolution of building details. The resolution, which is determined by generating a linearly-spaced grid, is designed by the number of cells in each axis. Once the linearly-spaced cells are defined, the occupied cells based on the transformed point cloud data in the principal coordinate system are specified. Figure 8 shows examples of the grid generation. Figure 8(a) is a low-resolution case with a single cell, and Figure 8(b) is a high-resolution case with multiple cells.



(a) Low-resolution (a single cell) (b) High-resolution (multiple cells)

Fig. 8: Results of grid generation

Using the grid definition, the height of each individual grid cell is computed. Simply, the height is computed by estimating the average value of all the points located in each cell. Based on the results of length, width, and height in each cell, the cuboid shapes of all the cells can be fully defined. Because the cuboid definition is in the principal coordinate system, it requires a coordinate transform from the principal axes system to the global coordinate system using Equation 5. Figure 9 shows the results of a building model. Figure 9(a) is the actual shape using the Delaunay triangulation of the point cloud. Figure 9(b) is the urban modeling result using the single cell approach, and Figure 9(c) is the result of the multiple cells approach. The example studies show that a higher number of cells leads to a higher resolution building. To generate an entire urban environment, this process is repeated until we create building models for all of the clusters.



(a) Example of a building (b) Result of a building model with a single cell (Low fidelity model) (c) Result of a building model with multiple cells (High fidelity model)

Fig. 9: Grid generation results in three dimensional space

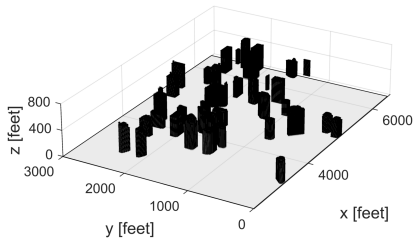
E. Examples of urban construction

In this section, we demonstrate the proposed rapid, data-driven, and grid-based urban modeling methodology. The accuracy quantification of the urban models is not feasible since we do not have the actual measurements of the selected cities. Other literature sources have also reported the qualitative analysis instead of a quantitative evaluation for the proposed urban modeling accuracy because of the lack of actual measurements [6][21]. For the comparison of the different fidelities, we select single-grid and multiple-grid approaches, and construct the San Diego urban model. Figures 10 are the results of the urban modeling using multiple- and single-grid approaches. The qualitative visual inspection shows that both approaches can successfully construct the buildings that are higher than the altitude constraint.

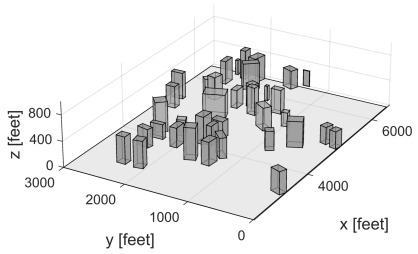
Using the proposed urban modeling methodology, we generate seven different cities shown in Figure 11. The results show that the buildings of all the seven cities are successfully detected and characterized precisely. The results also show that this technique could generate an urban model in an automated manner. This proposed LiDAR source-based urban modeling method has been successfully applied to test a UAV collision avoidance algorithm in the literature [9].

III. Artificial Urban Modeling Method

Although the introduced rapid, data-driven and grid-based urban modeling method can generate realistic urban models with a different level of fidelity, it has some limitations. The required airborne LiDAR may not exist or may be too expensive to acquire. To overcome these limitations, this paper introduces an alternative approach that does not rely on any LiDAR information. The alternative approach generates an artificial city model with similar characteristics of an interesting urban area, but its outcome has different locations and shapes of buildings. The main idea of this method is applying key metrics that is number of buildings, minimum



(a) Result of dense urban model with multiple grid cells



(b) Result of dense urban model with a single cell

Fig. 10: Example of urban models

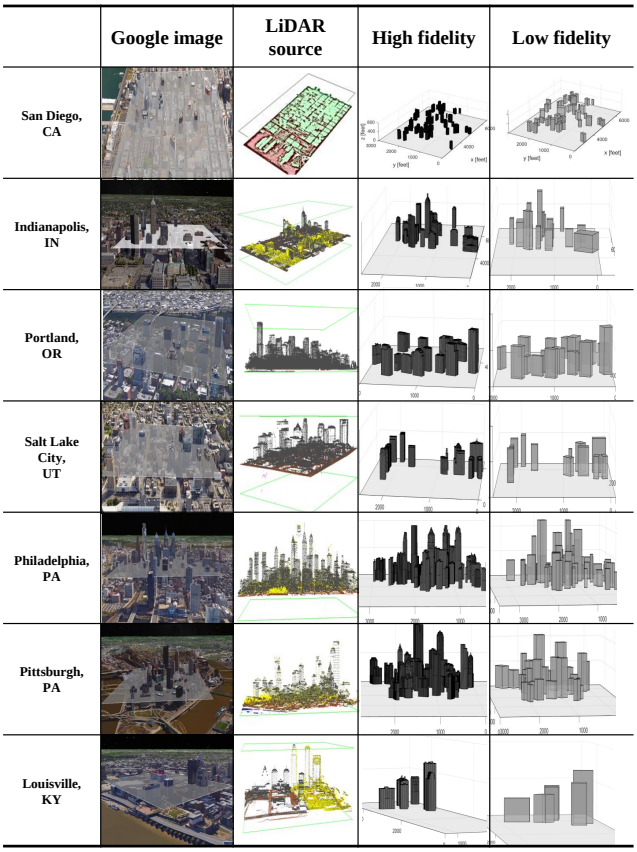


Fig. 11: Examples of realistic urban environments

separation distance, and the size of an interesting urban area. These metrics are reference inputs to determine the shapes and locations of buildings instead of using the actual LiDAR information. We note that the minimum separation distance is the shortest among the distances between all combination of buildings. The proposed artificial urban modeling method includes two main steps: adaptive spacing and force balancing. The following subsection describes two main steps with simple examples, and then introduces the proposed artificial urban modeling method with an actual implementation.

A. Adaptive spacing and force balancing algorithms

The artificial urban modeling method is an iterative approach. The initial positions of buildings are randomly defined within the boundary constraints. Based on the initial locations of the nodes, the connectivities between nodes are specified, which is called *adaptive spacing*. Based on the topological graph with the nodes and the connectivities, the algorithm solves a force balance equation that defines the internal forces of the flexible bars, connected edges, to reach target lengths, which is called *force balancing*.

The adaptive spacing algorithm minimizes errors, the difference between the lengths of edges and target lengths. The core idea of the adaptive spacing technique is inspired from an adaptive mesh refinement technique that has been implemented in a finite element method. Persson et. al have introduced an iterative mesh regeneration technique through solving a force equilibrium equation to refine meshes [22]. Conceptually, the adaptive mesh refinement technique applies the Delaunay triangulation method to generate meshes [23]. In each iteration, it solves a force-equilibrium equation that updates the new locations of nodes. This process is repeated until all the nodes reach the equilibrium states. This adaptive mesh refinement algorithm is simple, and creates robust and high quality meshes. Based on Persson's adaptive mesh refinement method, many other techniques have been introduced to handle more complex mesh refinement techniques [24] [25].

Our proposed adaptive spacing method is slightly modified from the adaptive mesh algorithm suggested by Persson et. al. The adaptive spacing algorithm utilizes the connectivity configuration resulting from the connectivity of nodes instead of applying the Delaunay triangulation method in Persson's adaptive mesh refinement technique [22]. The connectivity is defined by the closest node in the given topology. This connecting closest node may create multiple networks instead of one network. To circumvent this, if the closest node is already connected with a node, the next closest node is connected with the node. For the identification of the connection between nodes, Euclidian distance is applied.

Figure 12 illustrates the free body diagram of an example to explain the adaptive spacing method. It is assumed that the example configuration includes four nodes and four edges. For simplicity, the nodes are assumed to be the centers of the buildings, and the building shapes are ignored. The edges between the nodes are specified from the nearest node. In the figure, $n_i = [x_i \;\; y_i], : \quad (i = 1, 2, 3, 4)$ are nodes, and F_{ij} are

forces computed by the global force-equilibrium equation. The force of ith node is defined as follows

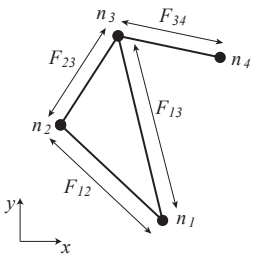


Fig. 12: Free body diagram of adaptive spacing

$$F_{n_i} = F_{internal,n_i} + F_{external,n_i}, \tag{6}$$

where $\mathbf{F}_{internal,n_i}$ is the internal force, which is a function of the edge length, and $F_{external,n_i}$ is the external force to satisfy the boundary constraint. The internal force $\mathbf{F}_{internal,n_i}$ is

$$\mathbf{F}_{internal,n_i} = [F_{n_i,x} \quad F_{n_i,y}] \tag{7}$$

where $F_{n_i,x}$ and $F_{n_i,y}$ denote ith forces in x and y direction, respectively. The equation of the force $F_{internal,n_i}$ is applied as a inear spring force defined as

$$F(l_j, l_{j,ref}) = \beta(l_{j,target} - l_j), \tag{8}$$

where β is a spring coefficient, $l_{j,target}$ is the target length of the jth bar, and l_j is the length of the jth bar. In Persson's model, the linear force equation only includes positive repulsive force, but not attractive force, since the final mesh refinement does not follow a mesh reference length [22][24] [25]. On the other hand, the adaptive spacing algorithm employs the force function with both repulsive and attractive forces so that each edge can converge to the desired length.

Based on the topology, the algorithm solves force-equilibrium equation to get bars' lengths to reach the target lengths $\mathbf{l}_{target}$. However, solving the force balance equation is not a simple problem since the forces are discontinuous functions resulting from the edge adaptation [22]. To solve the discontinuity issue of the force-equilibrium equation, we implement the following first-order ordinary differential equation with a pseudo-time step:

$$\frac{d\mathbf{n}}{dt} = \mathbf{F}(\mathbf{n}), \tag{9}$$

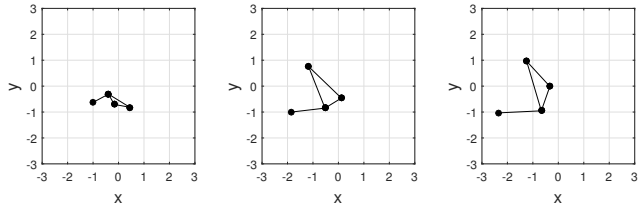
where the time t is artificial time, where $t \geq 0$. For the propagation of the equations of motion, we adopt a forward Euler method in the discrete time domain.

$$\mathbf{n}_{k+1} = \mathbf{n}_k + \Delta t \mathbf{F}(\mathbf{n}), \tag{10}$$

where Δt is the artificial time step and the initial locations of the nodes at time t_0 are $\mathbf{n}_0$.

The details of the adaptive spacing algorithm are summarized in Algorithm 1. In the algorithm, the minimum distances of all the nodes are computed, and the algorithm defines the connections between the nodes based on the result of the shortest distance. We note that to prevent sub-network clusters, once two nodes are already connected, the nodes with the next shortest distance are connected to each other. Based on the connectivity result, the force-equilibrium equation is solved, and the node positions are displaced according to the result of the ODE solver. This process is repeated until the solution converges. In every iteration the connectivity results can vary in response to the locations of nodes.

To test the performance of this algorithm, we conduct a numerical experiment with a simple example. The example has four nodes, and the target lengths of the four edges are $\mathbf{L}_0 = [1\,1.333\,1.666\,2]$. In this example, boundary constraints are not included, so the external force vector is $\mathbf{F}_{external} = \mathbf{0}$. Figure 13 shows the results of the numerical simulation in three different iterations. Figure 13(a) is the initial topology, randomly populated. The other two figures 13(b) and 13(c) are the 10th iteration, and the final iteration, respectively. Figure 14 is the response of the lengths of the four edges as the number of iterations increases. The solid lines are the target lengths, and the dashed lines are the responses of the edges' lengths. The iteration response clearly shows that all the lengths of the four edges converge to the target lengths after 50th iterations.



(a) Initial configuration (b) Configuration at the 10th iteration (c) Final configuration

Fig. 13: Example of adaptive edge

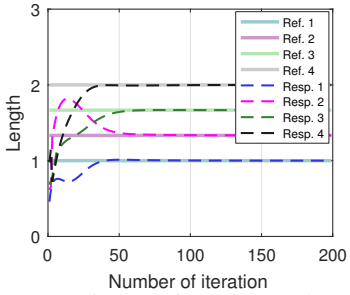


Fig. 14: Iteration profile of the edge lengths

In the urban modeling problem, all buildings must be located within a boundary so that the generated urban model can obtain the same density as the reference city. The boundary constraint is imposed by a penalty function, which has been applied to solve constrained optimization problems [26]. The penalty function converts a constrained problem into an unconstrained problem through introducing an additional term as a penalty function. There are several conventional penalty methods: exterior penalty function, an interior penalty function, and an extended interior penalty function methods. The exterior penalty function includes a quadratic penalty term that drastically increases the cost when the constraint is violated. The drawback of this method

is that the final solution may not be in a feasible region. On the other hand, the interior penalty function always produces a feasible solution. However, the initial starting point must be in a feasible region. The extended interior penalty function method combines the benefits of the other methods. The solution of the extended interior method may be violating a constraint, but the violation rate is less than the exterior penalty function. Moreover, the initial starting point does not need to be in the feasible area. Because of these benefits of the extended interior penalty function method, this paper implements this method to solve the boundary constraint problem. To be more specific, using the extended interior penalty function, the external forces help violated nodes to keep moving back to the feasible region [26]. The external forces resulting from the extended interior penalty function can be written as

$$\mathbf{F}_{external,i} = \begin{cases} -r_i \frac{1}{g_m(\mathbf{n_i})}, & if \quad g_m(\mathbf{n_i}) \leq \epsilon \\ -r_e \frac{2\epsilon - g_m(\mathbf{n_i})}{\epsilon}, & if \quad g_m(\mathbf{n_i}) > \epsilon, \end{cases} \tag{11}$$

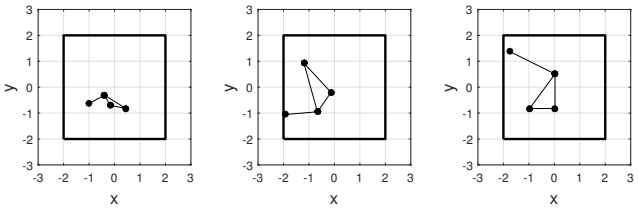
where $\mathbf{F}_{external,i}$ is the external force on the ith node. r_i and r_e are multipliers for the internal and external penalty functions that adjust the magnitudes of each force. In the interior penalty function, the transition parameter is ϵ, which is commonly a small negative number. The boundary constraints $g_m(\mathbf{n_i})$ can be defined by

$$\begin{aligned} g_1(\mathbf{n}_i) &= -c_{xmin} + n_{xi} \\ g_2(\mathbf{n}_i) &= c_{xmax} - n_{xi} \\ g_3(\mathbf{n}_i) &= -c_{ymin} + n_{yi} \\ g_4(\mathbf{n}_i) &= c_{ymax} - n_{yi}, \end{aligned} \tag{12}$$

where c_{xmin} and c_{xmax} are the minimum and maximum values of the boundary constraints in the x coordinate system, and c_{ymin} and c_{ymax} are the minimum and maximum values of the boundary constraint in the y coordinate system. To test the external penalty function, we conduct the numerical simulation using the previous four nodes example. In the example, it is assumed that the boundary constraints are $[c_{xmin} \;\; c_{xmax} \;\; c_{ymin} \;\; c_{ymax}] = [-2 \;\; 2 \;\; -2 \;\; 2]$. Figure 15 is the result of the adaptive spacing algorithm with boundary constraints. The results show that the final locations of all the nodes are within the boundary constraints. Figure 16 shows more details of the node changes according to the iteration number, these results show that the lengths of all four edges converge to the target lengths after 100 iterations. Compared to the adaptive spacing algorithm without a boundary constraint, the adaptive spacing algorithm with the boundary constraints requires more iterations because of the penalty function. The result also reveals that unlike the unconstrained adaptive spacing algorithm, the responses of all the edge lengths yield discontinuous trends on the 34th iteration because the external force by the boundary violation changes all the connectivities.

B. Urban modeling method using adaptive spacing

The urban modeling model using the adaptive spacing technique has a similar procedure to Algorithm 1. The main



(a) Initial configuration (b) Configuration at the 10th iteration (c) Final configuration

Fig. 15: Example of adaptive spacing algorithm with boundary constraints

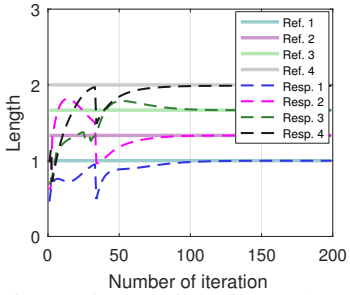


Fig. 16: Iteration profile of the edge lengths

difference of the urban modeling method is that the adaptive spacing algorithm measures the distance from a node, while the urban modeling problem computes the minimum distance from all four boundaries of a building. Algorithm 2 summarizes the pseudo-code of the proposed adaptive spacing-based urban modeling method.

In the algorithm, the input variables are the number of buildings, building characteristics (length/width/height/rotational angle), and boundary constraints. The number of buildings and the building characteristics are defined from the distribution of a given city. In this paper, the distribution is computed from the result of the urban model shown in Figure 11. Based on the distribution, the algorithm generates random numbers to define the features of all the buildings.

Figure 17 is an example of the distributions of all the buildings in San Diego. It is assumed that the distribution

Algorithm 1 Adaptive edging method

Inputs:
Number of nodes N_n
Artificial time step Δt
Penalty parameters r_i, r_e, ϵ
Target length vector $\mathbf{L}_0$
Boundary constraint $\mathbf{g}(x, y) \leq 0$
Initialization: Randomly populate all nodes
Initial location of nodes $\mathbf{n} = [n_x \;\; n_y]$, $(n_x, n_y \in R^{N_n})$
1. Compute Euclidian distances with other nodes
2. Connect nodes with the shortest distance,
   if the edges of two nodes are not connected to each other
3. Compute internal force, external force and total force according to Equations 7, 11, and 6
4. Forward propagate ODE function according to Equation 10 Converge

function is to be the gamma distribution, and the rotational angle is the uniform distribution based on the initial observation of the data. Using these distributions, we generate random numbers to determine the characteristics of all the buildings. The minimum separation distance is also determined by the distribution. Figure 17(d) is the distribution of the minimum separation distance in San Diego. From the result of the fitted distribution, the minimum separation distances, which are the target lengths $\mathbf{L}_0$ ($\mathbf{L}_0 \in R^{N_n}$), are randomly generated. The total number of buildings, N_n is 37. The boundary constraints are determined from the boundaries of all the buildings in an area of interest.

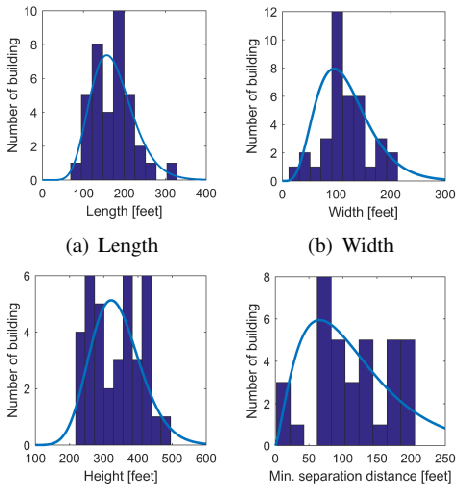
In the initialization phase, the initial locations of all the building are identified. The centers of the initial buildings are randomly defined within the boundary constraints. Then, the adaptive spacing algorithm is applied. To be more specific, based on the target distances, which are the minimum separation distances, it defines a new connectivity configuration, and then solves the force balance equation. Using the results of the force balance equation, the algorithm performs the forward propagation based on the first order differential equation. This process is repeated until it converges.

Algorithm 2 Urban modeling method using adaptive edging method

Inputs:
Number of buildings N_n
Length, width, height of building, and rotational angle L_b, W_b, H_b, θ
Artificial time step Δt
Penalty parameters r_i, r_e, ϵ
Target length vector $\mathbf{L}_0$
Boundary constraint $\mathbf{g}(x, y) \leq 0$
Initialization: Randomly populate the centers of all buildings
Initial location of nodes $\mathbf{n} = [n_x \quad n_y]$, $(n_x, n_y \in R^{N_n})$
1. Compute distances with other buildings
2. Connect the edges of the buildings
   with a shortest distance, if the edges of two
   buildings are not connected to each other
3. Compute internal force, external force and total
   force according to Equations 7, 11, and 6
4. Forward propagate ODE function according to Equation 10 Converge

To test the proposed urban modeling algorithm with the adaptive spacing technique, we select downtown San Diego as an example study. Figure 18 shows the results of the urban modeling method indicating the changes of the building connectivities as the number of iterations increases. In the initial urban configuration, all the buildings are close together because we randomly populate all buildings around the center of the boundary constraints. As the iteration number increases, all the buildings are distributed over the entire urban area.

Figure 20 depicts the distribution changes according to the number of iterations. The results illustrate that the

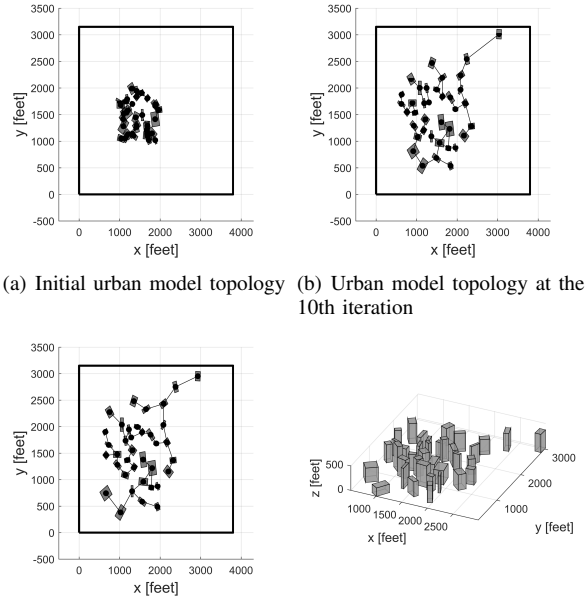


(a) Length (b) Width

(c) Height (d) Min. separation length

Fig. 17: Reference distributions

initial distribution shape is significantly different from the target distribution, but as the iteration number increases, the distribution is close to the reference distribution. To compare the two urban models (actual vs. artificial), the bar charts are displaced in Figure 20. As expected, the two histograms are slightly different, but the fitted distributions between the two histograms are similar. In conclusion, the proposed adaptive algorithm can generate an artificial urban area that is visually different, but has a similar distribution of the minimum separation distance.



(a) Initial urban model topology (b) Urban model topology at the 10th iteration

(c) Urban model topology at the 200th iteration (d) Final urban model topology

Fig. 18: Urban modeling results

IV. Conclusion

Simulations for testing a UAV collision avoidance algorithm use simple urban models featuring cuboid or cylindrical shapes. Most of these models did not report the relevance to the actual urban environment and introduce a

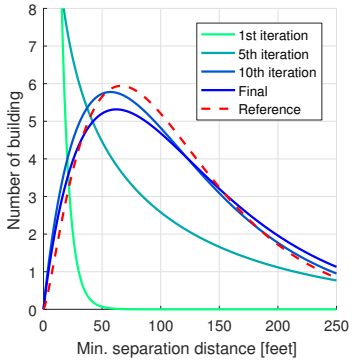
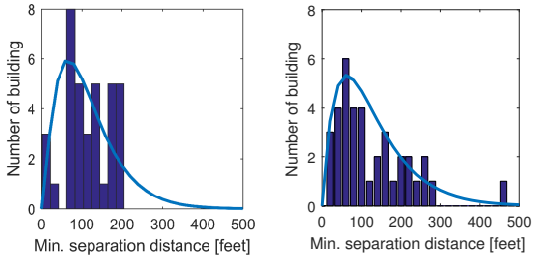


Fig. 19: Distribution changes as the number of iterations increases



(a) Distribution of San Diego (b) Distribution of the artificial urban model

Fig. 20: Distribution comparison

urban modeling process. To address these problems, this paper introduces two urban modeling methods: a rapid, data-driven, and grid-based urban modeling method, and an artificial urban modeling method.

The rapid, data-driven and grid-based urban modeling method is an airborne LiDAR information-based approach. Using the LiDAR sensor information, this method introduces five steps to generate an urban model: data resampling, clustering, principal component analysis, grid-generation, and construction of a building. Numerical simulations are conducted to qualitatively assess the results of the proposed urban modeling method. The results of the seven cities show that the method successfully generates realistic urban models and adjusts their fidelity based on the grid resolution.

The artificial urban modeling method can generate a reference city without any LiDAR resources. This technique uses the characteristics of urban models to determine the references (i.e., urban characteristics, and minimum distance between buildings) and applies the adaptive spacing method to populate buildings. We demonstrated this artificial urban modeling method using downtown the San Diego model. Results of the proposed urban modeling method reveal that it successfully generates an artificial San Diego model that has a different shape, but similar characteristics.

REFERENCES

[1] N. Xu, G. Cai, W. Kang, and B. M. Chen, "Minimum-time trajectory planning for helicopter uavs using computational dynamic optimization," in *In Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2732 – 2737.

[2] N. Wen, L. Zhao, X. Su, and P. Ma, "UAV online path planning algorithm in a low altitude dangerous environment," *Automatica Sinica*, vol. 2, no. 2, 2015.

[3] S. Griffiths, J. Saunders, A. Curtis, B. Barber, T. McLain, and R. Beard, "Obstacle and terrain avoidance for miniature aerial vehicles," *In Advances in Unmanned Aerial Vehicles*, 2007, springer Netherlands.

[4] T. J. Stastny, G. A. Garcia, and S. S. Keshmiri., "Collision and obstacle avoidance in unmanned aerial systems using morphing potential field navigation and nonlinear model predictive control," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 1, 2015.

[5] H. Jinhui, S. Youand, and U. Neumann, "Approaches to large-scale urban modeling," *Computer Graphics and Applications*, vol. 23, no. 6, 2003.

[6] J. Hu, S. You, U. Neumann, and K. K. Park, "Building modeling from LiDAR and aerial imagery," in *Proceedings of ASPRS*, 2004.

[7] S. You, J. Hu, U. Neumann, and P. Fox, "Urban site modeling from LiDAR," in *ICCSA'03 Proceedings of the 2003 international conference on Computational science and its applications*, 2003, pp. 579 – 588.

[8] Q.-Y. Zhou and U. Neumann, "Fast and extensible building modeling from airborne LiDAR data," in *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2008.

[9] Y. Choi, H. Jimenez, and D. N. Mavris, "Two-layer obstacle collision avoidance with machine learning for more energy-efficient unmanned aircraft trajectories," *Robotics and Autonomous Systems*, vol. 98, pp. 158–173, 2017.

[10] Y. Choi, "A framework for modeling and simulation of control, navigation, and surveillance for unmanned aircraft separation assurance," Ph.D. dissertation, Georgia Institute of Technology, 2016.

[11] "UTM: Air traffic management for low-altitude drones," NASA, Tech. Rep., 2015.

[12] J. A. Hartigan and M. A. Wong., "Algorithm as 136: A k-means clustering algorithm." *Journal of the Royal Statistical Society*, vol. 28, no. 1, pp. 100 – 108, 1979.

[13] D. T. Pham, S. S. Dimov, and C. D. Nguyen, "Selection of k in k-means clustering," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 219, no. 1, pp. 103 – 119, 2005.

[14] H. Jia, S. Ding, X. Xu, and R. Nie, "The latest research progress on spectral clustering," *Neural Computing and Applications*, vol. 24, pp. 1477–1486, 2014.

[15] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing system*, vol. 2, pp. 849 – 856, 2002.

[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, Eds., *A density-based algorithm for discovering clusters in large spatial databases with noise*, vol. 96, no. 34. International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996.

[17] S. Ghosh, D. Rancourt, and D. N. Mavris, "Principal component analysis assisted surrogate modeling (PCA-SM) of correlated loads for uncertainty analysis of design load envelopes," in *16th AIAA/ISSMO Multidisciplinary analysis and optimization conference*, 2016.

[18] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[19] I. Alexander and T. Raiko, "Practical approaches to principal component analysis in the presence of missing values," *The Journal of Machine Learning Research*, vol. 11, pp. 1957 – 2000, 2000.

[20] I. Jolliffe, *Principal component analysis*. John Wiley & Sons, 2002.

[21] V. Verma, K. Rakesh, and S. Hsu, "3D building detection and modeling from aerial LIDAR data," in *2006 IEEE Computer Society Conference*, vol. 2, 2006, pp. 2213 – 2220.

[22] P.-O. Persson and G. Stran, "A simple mesh generator in MATLAB," *SIAM review*, vol. 46, no. 2, pp. 329 – 345, 2004.

[23] H. Edelsbrunner, *Geometry and topology for mesh generation*. Cambridge University, 2001.

[24] R. Holm, R. Kaufmann, B. Heimsund, E. ian, and M. Espedal, "Meshing of domains with complex internal geometries," *Numerical Linear Algebra with Applications*, vol. 13, no. 9, pp. 717 – 731, 2006.

[25] S. Li, Z. Xu, G. Ma, and W. Yang, "An adaptive mesh refinement method for a medium with discrete fracture network: The enriched Perssons method," *Finite Elements in Analysis and Design*, vol. 86, pp. 41 – 50, 2014.

[26] G. N. Vanderplaats, *Multidiscipline Design Optimization*. Vanderplaats Research & Development, Inc., 2007.